Programmeerimise põhikursus

ITI0010

Loengu ülevaade

Mängupuud ja mängude programmeerimine

Sissejuhatus viimasesse laboratoorsesse töösse: briti (ameerika) kabe

Mõtlemis-võistlusmängude programmeerimine

Mis on ülesanne?

Meil on lauamäng (male, kabe vms), kus on kaks mängijat.

Kumbki tahab võita.

Teeme arvutiprogrammi, mis võtab seisu ette ja püüab leida võimalikult hea käigu sellest seisust.

Veidi ajalugu

Teise maailmasõja paiku:

Alan Turing tegi valmis maleprogrammi, aga tal ei olnud arvutit selle täitmiseks. Turing täitis programmi ise: paberi ja pliiatsi abil.

Mängis paar partiid: iga partii võttis hirmus kaua aega.

1940-1990:

Programmid läksid aeglaselt paremaks, arvutid kiiremaks, aga tipp-inim-mängijad olid programmidest kõvasti paremad.

1990-2000:

Programmid hakkasid tipp-inim-mängijaid järjepanu võitma

CHINOOK – 1994, 8x8 inglise kabe maailmameister

Deep Blue – 1997, võitis Garry Kasparovit malematshis

LOGISTELLO – 1997, võitis Othello (Reversi) maailmameistrit Takeshi Murakamit

Praegune seis:

Mitmete mängude (Go, Bridzh, ...) jaoks on inim-mängijad endiselt programmidest hulga paremad

Kuidas maleprogrammi teha?

Teeme kõigepealt programmi, mis:

- loeb seisu
- teeb seisu järgi mällu tabeli kõigist võimalikest käikudest selles seisus.

See ei olegi nii väga keeruline programm

- Males keerulisem
- Kabes lihtsam
- Othello (reversi) veel lihtsam
- Viis nuppu ritta: väga lihtne

Nüüd võib programm tabelist juhusliku käigu valida.

Ja juba mängibki! Kuigi kehvasti ....

Kuidas programmi veidi paremaks teha?

Teeme eraldi väikese programmi, mis:

Võtab seisu ette
Ütleb umbes, kui hea seis on.
Seisu “headus” on number, mille meie programm arvutab.

Kuidas öelda “kui hea”?

Vahel lihtne: kaotus / võit/ ei kumbki.
Numbrid: 1 -1 0

Keerulisem: “harilik maleseis”. Kumbalgi pole võitu ega kaotust.

Seisu hindamine

Keerulisem olukord: harilik maleseis.

Materjal:

- Loeme kokku meie ja vastase nupud.
- Lipp annab 10 punkti, vanker 6, jne.

Positsioon:

- Loeme kokku eelviimasel real etturid masinal/inimesel
- Loeme kokku tsentriväljadel olevad etturid masinal/inimesel
- Loeme kokku, mitmele tsentriväljale masin/inimene tuld annab
- Vaatame üle kuninga kaitse tugevuse
- .... jne .....

Kokku saame seisu headuse numbri:

- Masina materjal – inimese materjal + masina positsioon – inimese positsioon

Mis edasi?

Nüüd on meil kõigi käikude tabel, iga käigu juures arvutame tekkiva seisu headuse numbri.

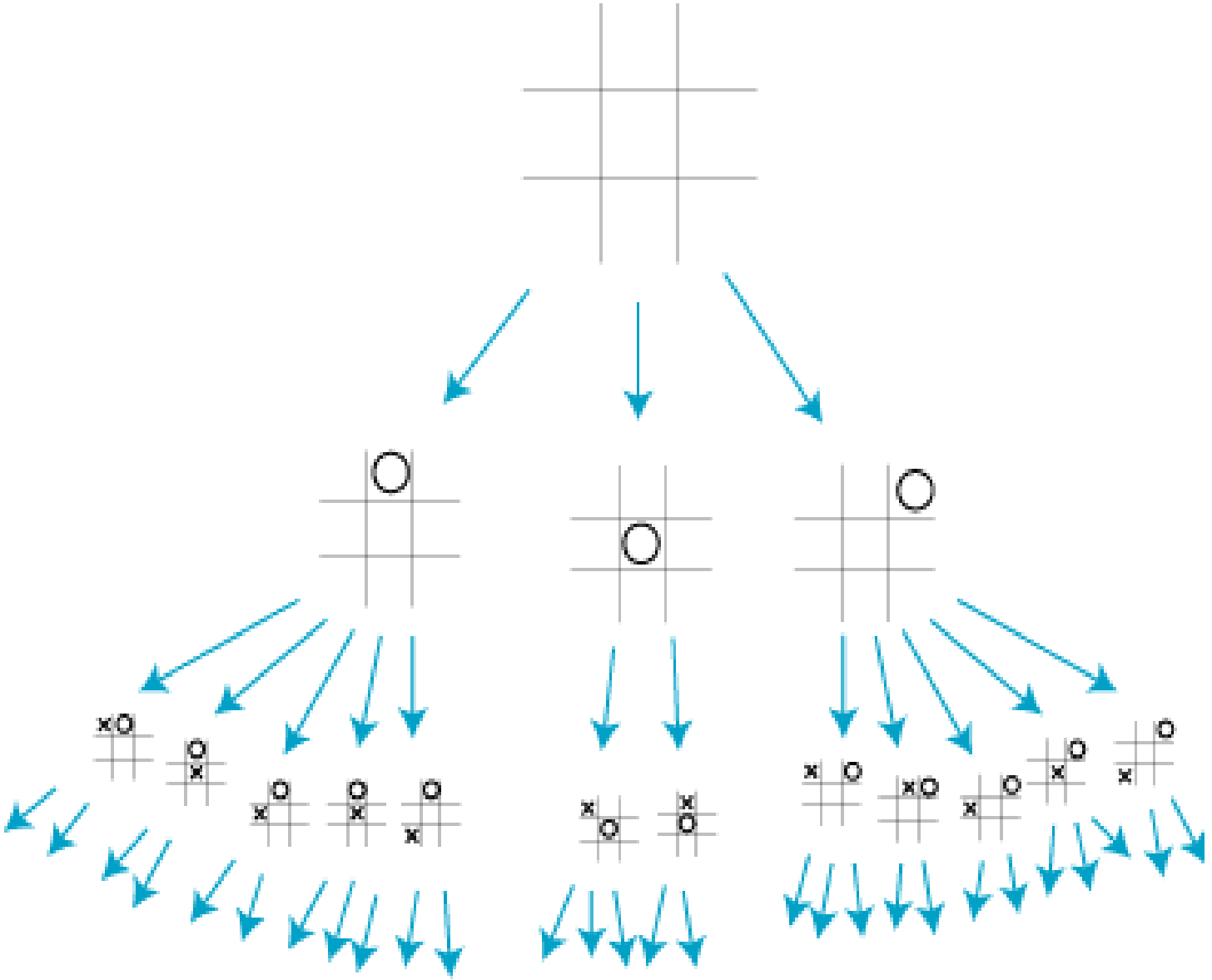
Valime kõige parema tulemuse (maksimumi numbritest!) andva käigu.

Programm hakkab veidi paremini mängima, aga:

- Ikka mängib väga kehvasti!
- Miks? Sest meie seisu headuse hindamise programm on nigel.
- Kuidas headuse hindamist parandada?

Loomulikult tuleks vaadata, mis käike vastane võib peale meie käike teha! Ja kuidas meie võime vastata. Jne.

Võimalike käikude puu trips-traps-trulli näitel



Kuidas sellist seisude puud kasutada?

Eeldame lihtsalt, et:

masin tahab teha käiku, mis annab kõige suurema headuse numbriga seisu.

vastane tahab teha käiku, mis annab kõige väiksema headuse numbriga seisu.

Seega:

igast seisust valib masin käigu, mis on maksimum-headusega (masinale).

igast seisust valib vastane käigu, mis on miinimum-headusega (masinale).

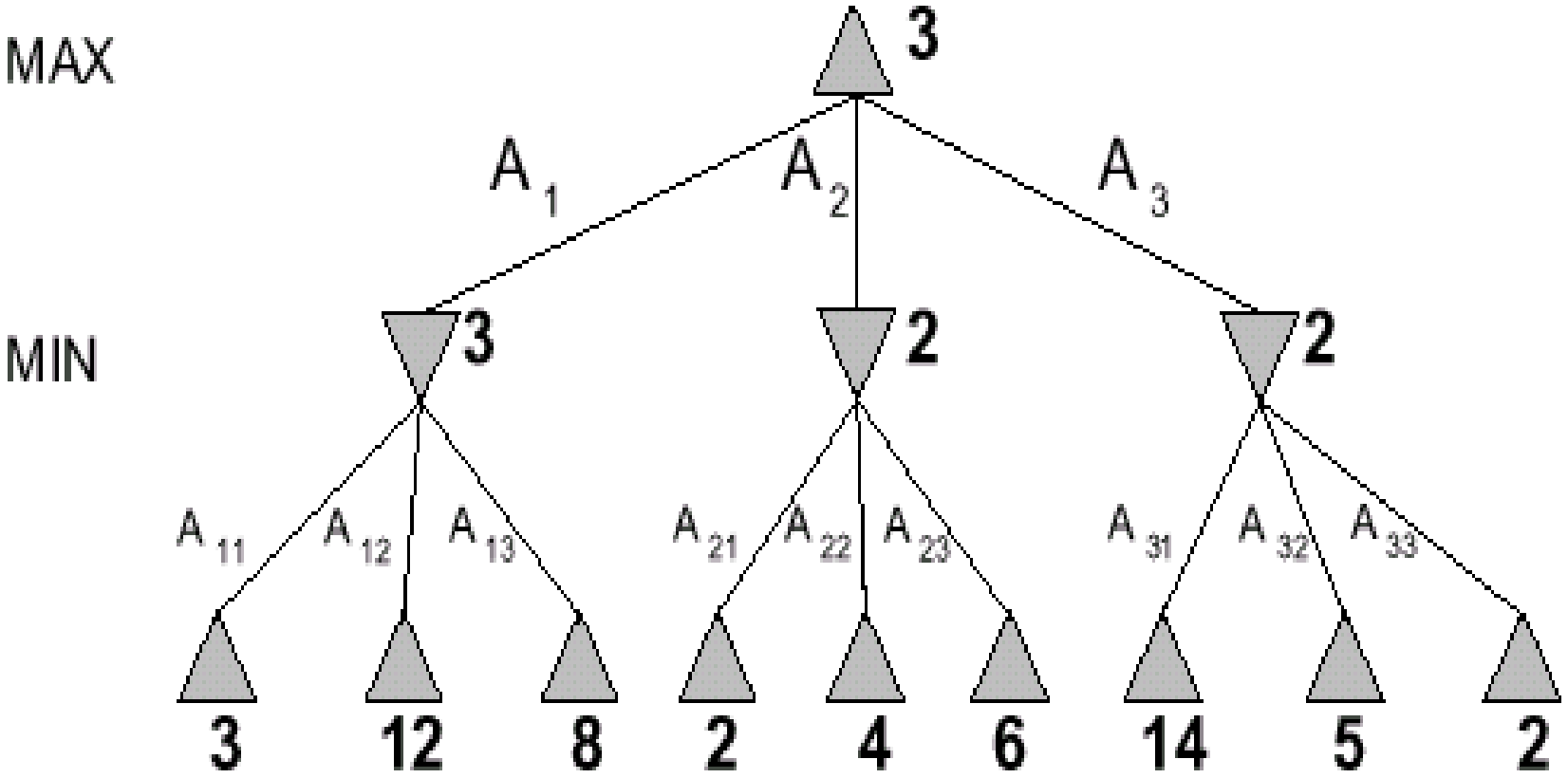
Idee:

Vaatame käikude puud sügavuseni N (näiteks N=3)

Kõige alumistel seisudel arvutame lihtsalt headuse välja

Seejärel “tõstame” headuse numbreid ülespoole!

Minimax algoritm



Vaata lisaks linki ja demo:
http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html

Kui heaks programmi saab?

Mida sügavamat puud masin läbi jõuab vaadata, seda täpsemini ta käiku oskab valida.

Puu läheb kiiresti väga suureks!

Males ca 30 käiku ühes seisus.

Esimesel tasemel käike 30.

Teisel tasemel käike 30*30

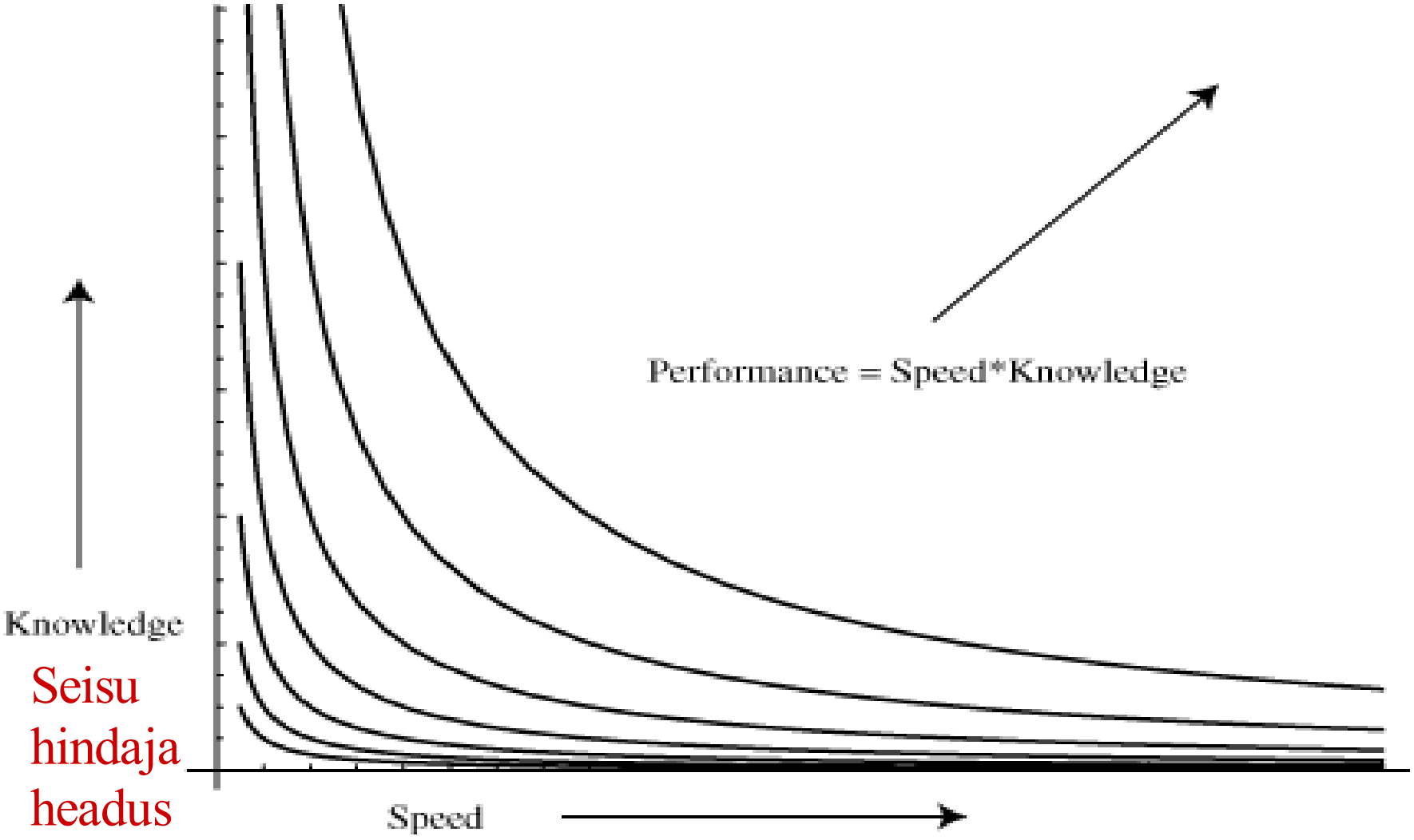
Kolmandal tasemel käike 30*30*30

....

N-ndal tasemel käike 30 astmes N.

Viiekümnendal tasemel oleks käike ca 30 astmes 50. See on rohkem, kui elementaarosakesi universumis!

Umbes nii: teadmised ja kiirus ja mängu headus



Seisu hindaja headus

Kui suure puu jõuab läbi vaadata

Kuidas programmi parandada?

Ei ole ühte head lahendust. On palju erinevaid nõkse!

Näiteks:

Teeme seisu hindaja paremaks (programmi “targemaks”).

Aga siis läheb ta aeglasemaks ka.

Seega jõuame vähem käike läbi vaadata.

Enamik nõkse on seotud käikude puu vähendamisega: ei ole vaja kogu puud läbi vaadata.

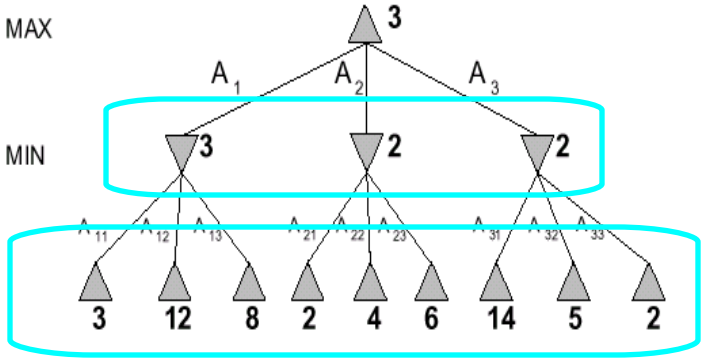
Laiuti vs sügavuti otsing

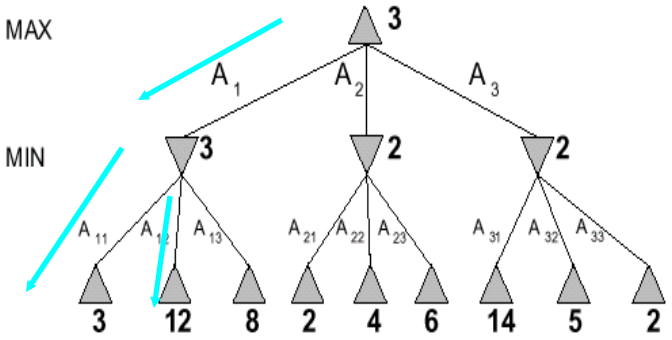
Kaks võimalust:

- Otsime puu läbi kiht-kihilt
- Otsime puu läbi sügavuti, minnes algul vasakul maksimaalse sügavuseni

Eelistatakse sügavuti otsingut!

- Mälu vaja palju vähem
- Muud eelised ka





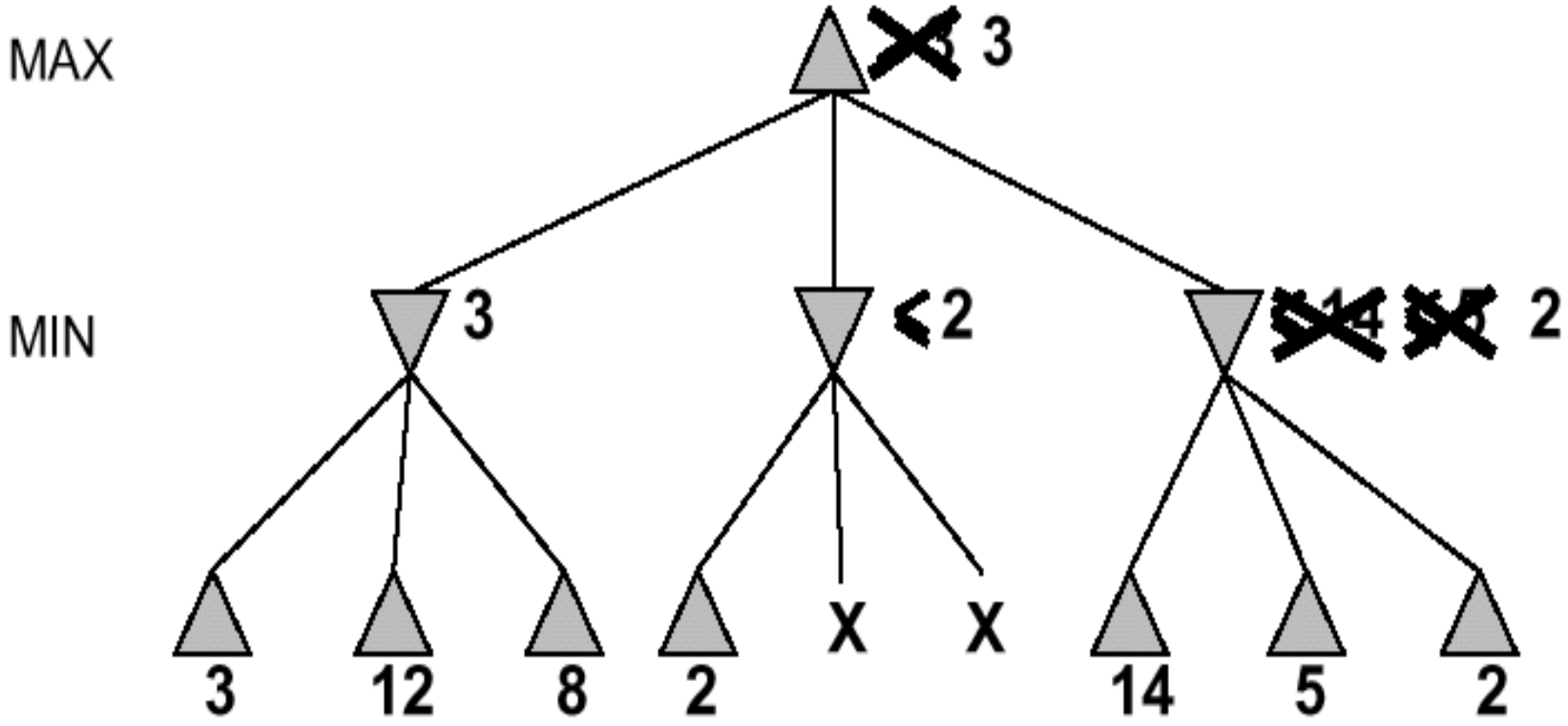
Koodinäiteid: rekursiivne minimax

```
int minimax(position p, int d) {
   struct move list[MAXMOVES];
   int i,n,bestvalue,value;

  if(checkwin(p)) {
     if (p.color==WHITE) return -INFINITY;
     else return INFINITY;
  }
  if(d==0) return evaluation(p);
  if(position.color==WHITE) bestvalue=-INFINITY;
  else bestvalue=INFINITY;
  n=makemovelist(p,list);
  if(n==0) return handlenomove(p);

  for(i=0;i<n;i++) {
    domove(list[i],&p);
    value=minimax(p,d-1);
    undomove(list[i],&p);
    if(position.color==WHITE)
       bestvalue=max(value,bestvalue);
     else
       bestvalue=min(value,bestvalue);
  }
  return bestvalue;
}
```

Üks esimesi universaalseid meetodeid: alpha-beta



Koodinäiteid: rekursiivsed negamax ja alpha-beta

```
int NegaMax (pos, depth)
{
    if (depth == 0) return Evaluate(pos);
    best = -INFINITY;
    succ = Successors(pos);
    while (not Empty(succ))
    {
        pos = RemoveOne(succ);
        value = -NegaMax(pos, depth-1);
        if (value > best) best = value;
    }
    return best;
}

int AlphaBeta (pos, depth, alpha, beta)
{
    if (depth == 0) return Evaluate(pos);
    best = -INFINITY;
    succ = Successors(pos);
    while (not Empty(succ) && best < beta)
    {
        pos = RemoveOne(succ);
        if (best > alpha) alpha = best;
        value = -AlphaBeta(pos, depth-1, -beta, -alpha);
        if (value > best) best = value;
    }
    return best;
}
```

Vt lisaks demo!

http://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html

Soteerimise nõks otsingu kiirendamiseks

Sorteerida variandid seisust X enne otsingut ära: alustada tõenäoliselt paremate käikude proovimisega.

Suurendab tohutult alpha-beta efekti!

Kuidas sorteerida?

Iteratiivne süvenemine. Teeme:

algul täisotsingu sügavuseni 2,

siis uue täisotsingu sügavuseni 4,

siis uue täisotsingu sügavuseni 6,

.... jne ....

iga kord kasutame eelmise otsingu tulemust sorteerimiseks!

Muud standardnõksud

"**Killer moves**": jätame otsides meelde eriti head käigud:

nii masinal kui vastasel

proovime kõigepealt varasemast meelde jäetud eriti häid käike

"**Quiescence search**": mõnda haru otsitakse sügavamalt:

ebastabiilses seisus otsime sügavamalt

stabiilses seisus otsime vähem sügavalt

otsime lõpuni kõik vahetused ja löögid

"Null-Move": mis siis, kui vastane saaks kaks käiku järjest?

Proovime nii, et vastane saab kaks käiku järjest

Kui on meile OK tulemus, siis see on positiivne faktor

Kui on meile halb tulemus, jätame meelde "killer move"

Kuidas mõjub?

Mida sügavam puu, seda suurem mõju. Näiteks:

Käikude puu sügavus viis:

MiniMax: hindab	10,541,242 seisu
Alpha-Beta: hindab	1,037,209 seisu
A-B + “killer moves”:	530,587 seisu.

Käikude puu sügavus seitse:

MiniMax: hindab ca	8,100,000,000 seisu
Alpha-Beta: hindab	162,662,568 seisu
A-B + “killer moves”:	46,455,262 seisu.

Täiendav idee: lõppmängude andmebaasid

Idee:

ehitame hiiglasliku lõppmängu-seisude andmebaasi

igal lõppmängu-seisul on andmebaasis öeldud täpne “headus” (võit, viik, kaotus)

ehitamiseks: kõigepealt ühe nupuga lõppmängud, siis kahe nupuga lõppmängud, siis kolme nupuga lõppmängud jne.

Kõigepealt tehti seda 8x8 inglise kabes:

Chinook: Jonathan Schaeffer, Robert Blake,
Paul Lu and Martin Bryant:
University of Alberta: http://www.cs.ualberta.ca/~chinook/

Kõigi seisude andmebaas, kus on
10 või vähem nuppu
Kokku üle **34,778,882,769,216** seisu

Sügavad otsingud jõuavad tihti lõppmängude andmebaasini, kus on juba sees täpne seisuhinnang.

Otsing seega mõlemast suunast!

Praegune seis mängudega

Lahendatud:

Neli nuppu ritta, Qubic,

Nine Man's Morris, Go-Moku

Awari

Tugevalt üle inimmängijate :

Kabe (8x8),

Renju (viis nuppu ritta),

Othello (Reversi), Scrabble, Backgammon

Maailmameistri tasemel :

Male,

Rahvusvaheline kabe (10x10)

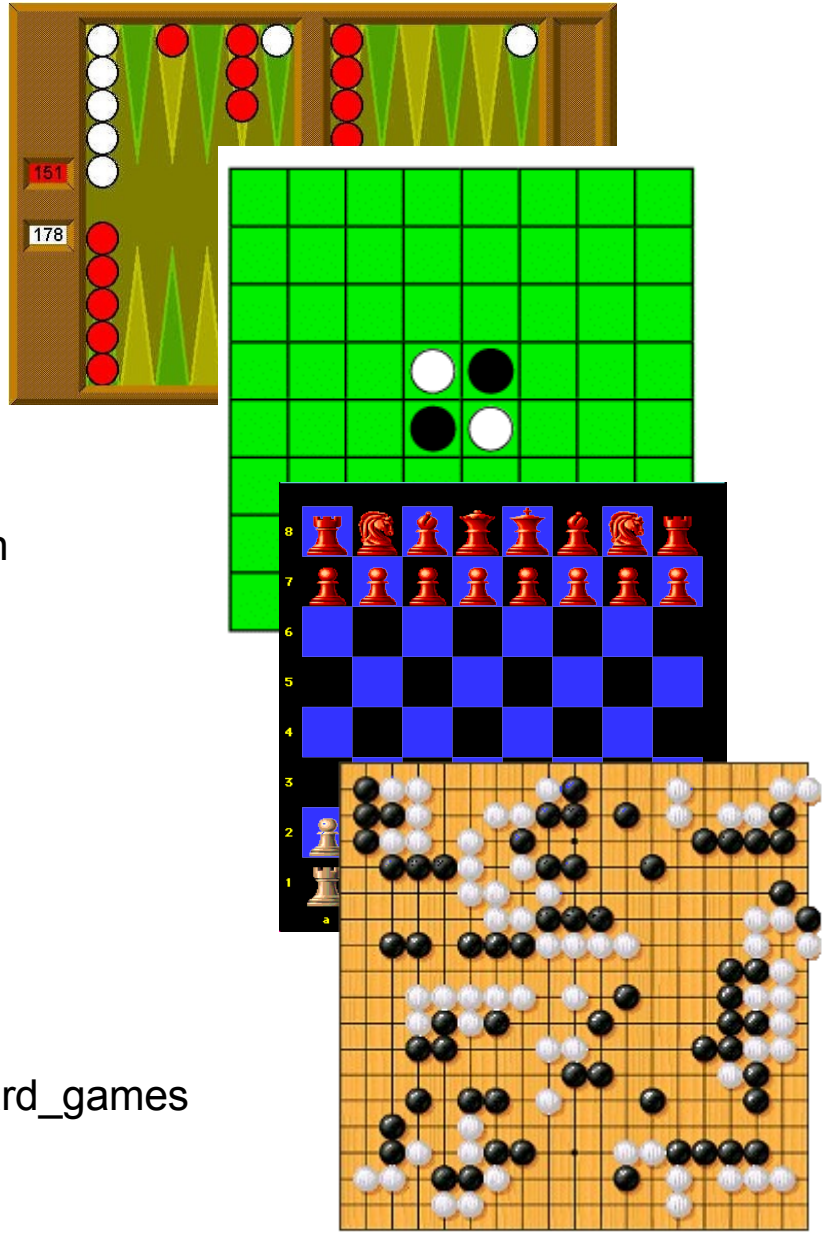
Tippmängija tasemel:

Go (9x9), Hiina male, Bridzh, Pokker

Amatööri tasemel :

Go (19x19)

Vt ka: http://en.wikipedia.org/wiki/Solved_board_games



Linke briti (ameerika) kabe kohta

Nimi: Checkers (draughts)

Reeglid: http://en.wikipedia.org/wiki/English_draughts ja üldiselt
http://en.wikipedia.org/wiki/Checkers

Baaskood kasutamiseks: õpikus lehel
http://math.hws.edu/javanotes/c8/s5.html toodud apleti osad, vt
http://math.hws.edu/javanotes/source/Checkers.java

Kasulikke linke lisaks eelmistele:
http://www.cs.ualberta.ca/~chinook/